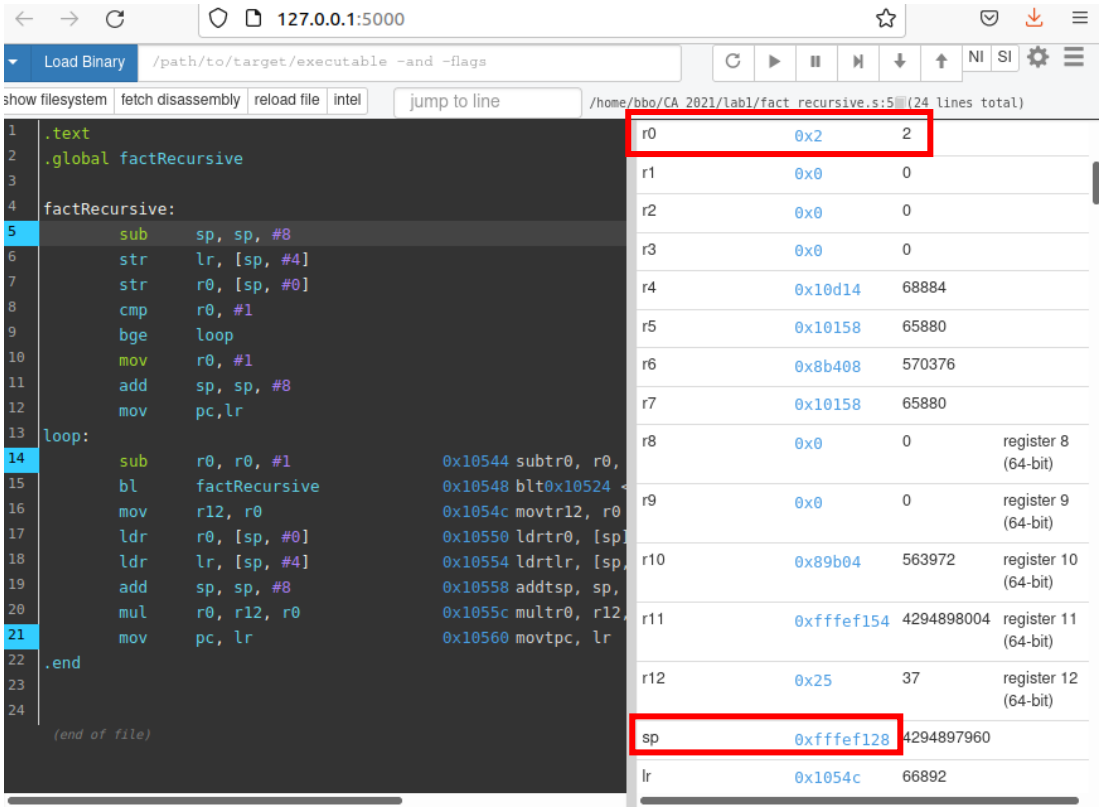
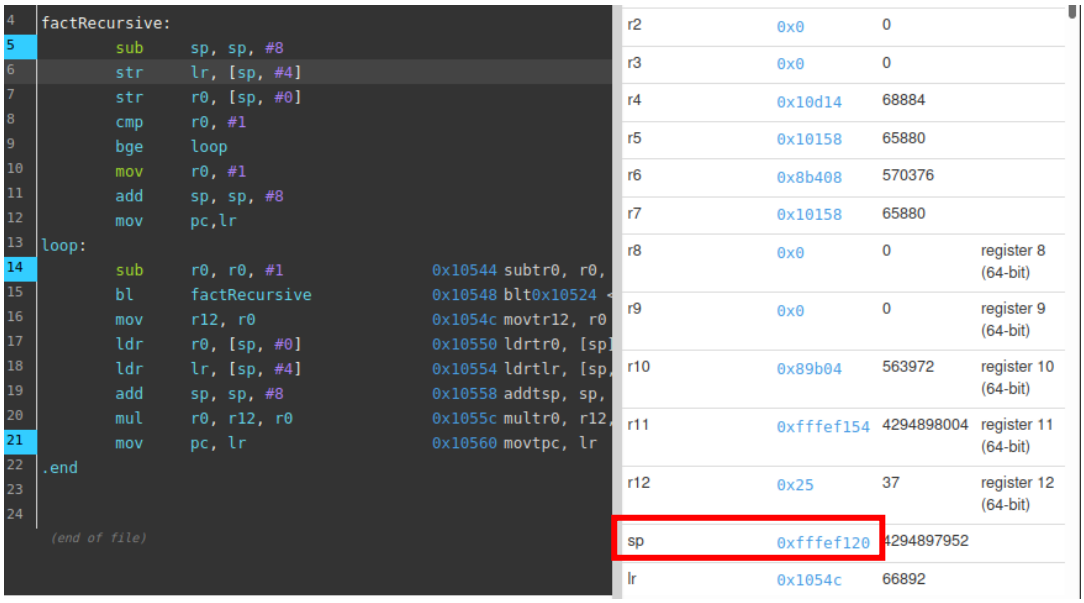
HW5 : factorial 함수 구현

20160394 임효상

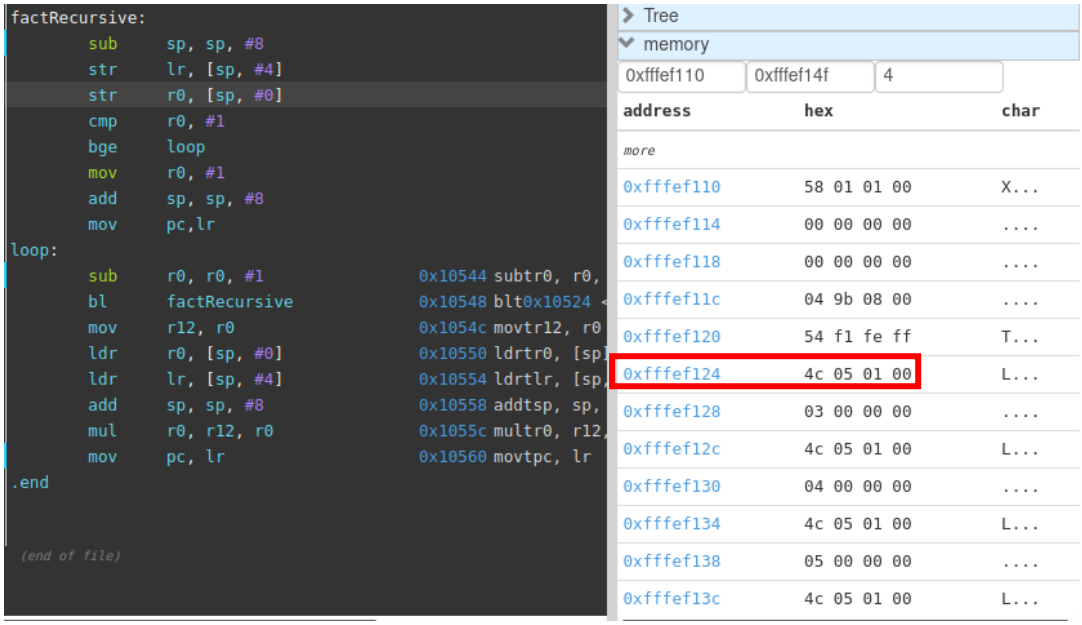
1) 실습 A



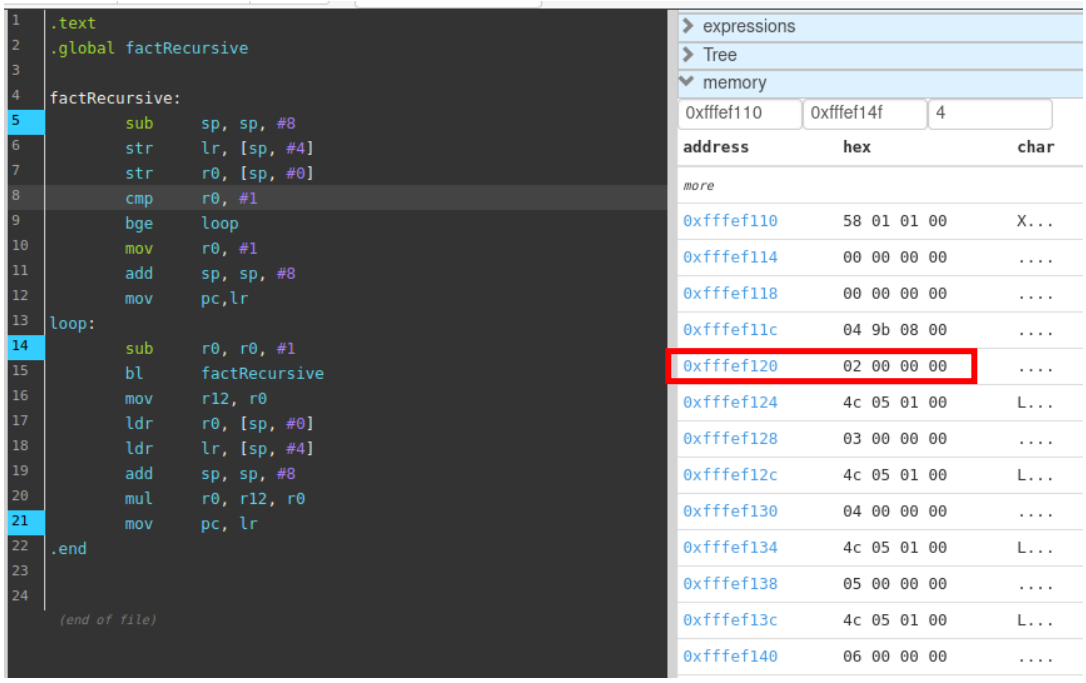
처음 r0 = 2, sp = 0xfffef128 이다.



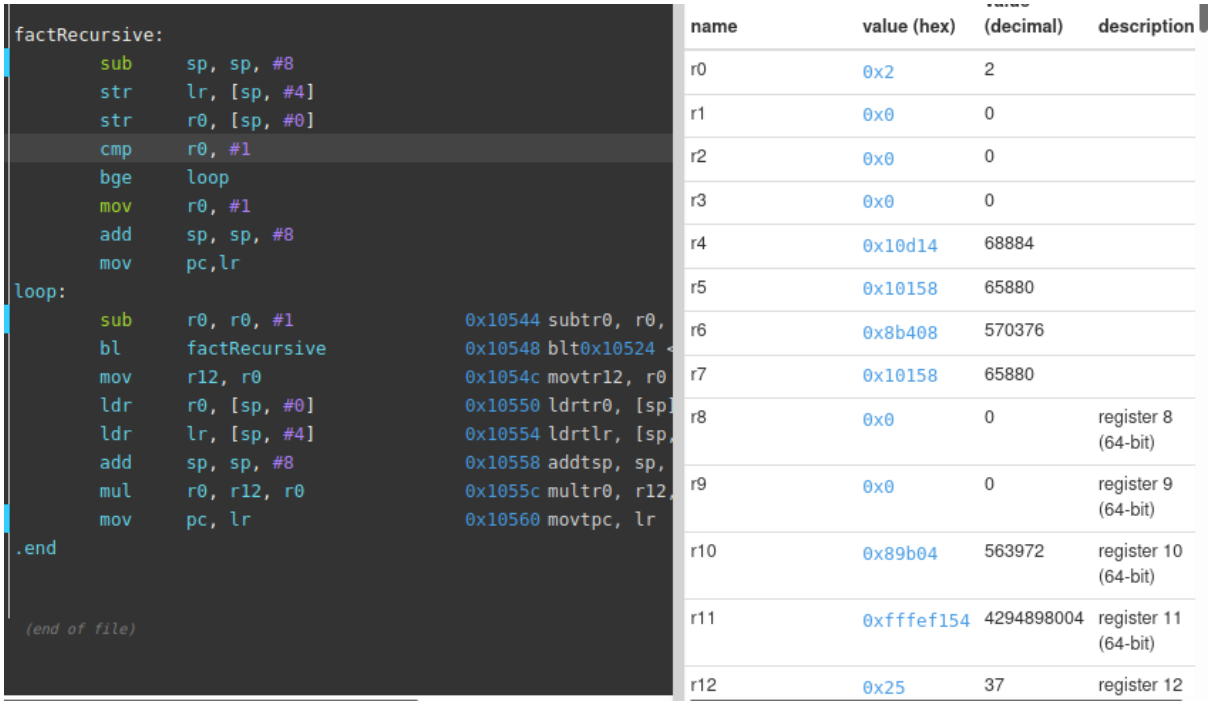
sub sp, sp, #8 명령어 수행 -> SUB명령어로 sp 값이 8만큼 감소되었다.



str lr, [sp, #4] 명령어 수행 -> 현재 sp값 + 4의 위치에 link register 값을 저장했다. link register의 값은 이전에 0x1054c로 저장되었다.



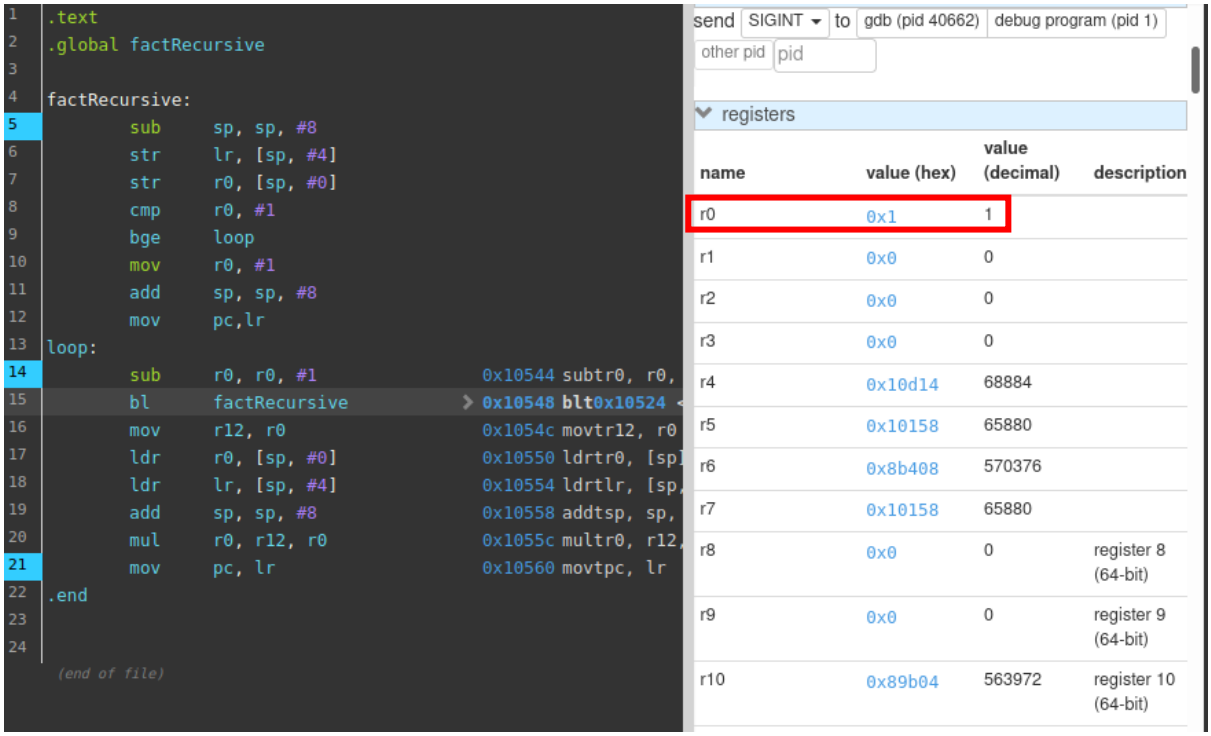
str lr, [sp, #0] 명령어 수행 -> 현재 sp값 위치에 r0(2)를 저장했다.



```
factRecursive:
        sub     sp, sp, #8
        str     lr, [sp, #4]
        str     r0, [sp, #0]
        cmp     r0, #1
        bge     loop
        mov     r0, #1
        add     sp, sp, #8
        mov     pc,lr
loop:
        sub     r0, r0, #1              0x10544 subtr0, r0,
        bl      factRecursive           0x10548 blt0x10524 <
        mov     r12, r0                 0x1054c movtr12, r0
        ldr     r0, [sp, #0]            0x10550 ldrtr0, [sp]
        ldr     lr, [sp, #4]            0x10554 ldrtlr, [sp,
        add     sp, sp, #8              0x10558 addtsp, sp,
        mul     r0, r12, r0             0x1055c multr0, r12,
        mov     pc, lr                  0x10560 movtpc, lr
.end

(end of file)
```

name	value (hex)	(decimal)	description
r0	0x2	2	
r1	0x0	0	
r2	0x0	0	
r3	0x0	0	
r4	0x10d14	68884	
r5	0x10158	65880	
r6	0x8b408	570376	
r7	0x10158	65880	
r8	0x0	0	register 8 (64-bit)
r9	0x0	0	register 9 (64-bit)
r10	0x89b04	563972	register 10 (64-bit)
r11	0xfffef154	4294898004	register 11 (64-bit)
r12	0x25	37	register 12

cmp r0, #1과 bge loop 명령어 수행 -> r0와 1의 값을 비교, r0값이 크므로 loop로 branch한다.



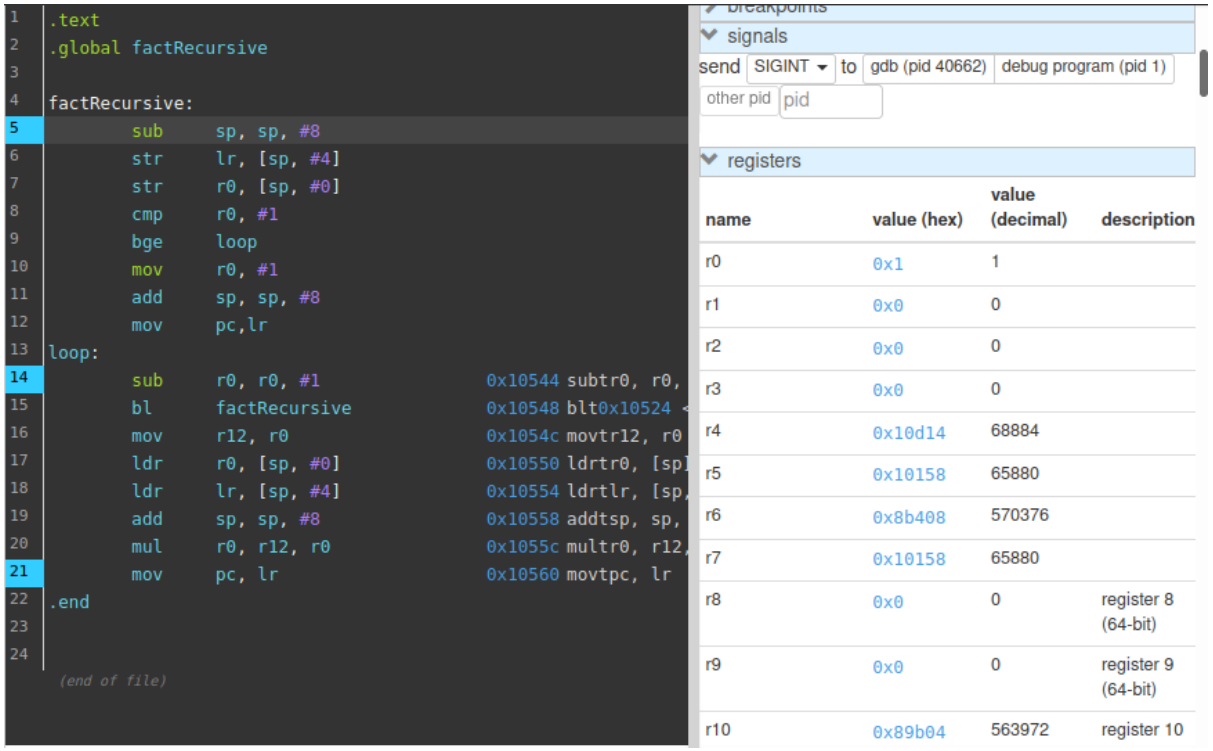
```
1  .text
2  .global factRecursive
3
4  factRecursive:
5          sub     sp, sp, #8
6          str     lr, [sp, #4]
7          str     r0, [sp, #0]
8          cmp     r0, #1
9          bge     loop
10         mov     r0, #1
11         add     sp, sp, #8
12         mov     pc,lr
13 loop:
14         sub     r0, r0, #1              0x10544 subtr0, r0,
15         bl      factRecursive           0x10548 blt0x10524 <
16         mov     r12, r0                 0x1054c movtr12, r0
17         ldr     r0, [sp, #0]            0x10550 ldrtr0, [sp]
18         ldr     lr, [sp, #4]            0x10554 ldrtlr, [sp,
19         add     sp, sp, #8              0x10558 addtsp, sp,
20         mul     r0, r12, r0             0x1055c multr0, r12,
21         mov     pc, lr                  0x10560 movtpc, lr
22 .end
23
24
   (end of file)
```

send SIGINT to gdb (pid 40662) debug program (pid 1)
other pid pid

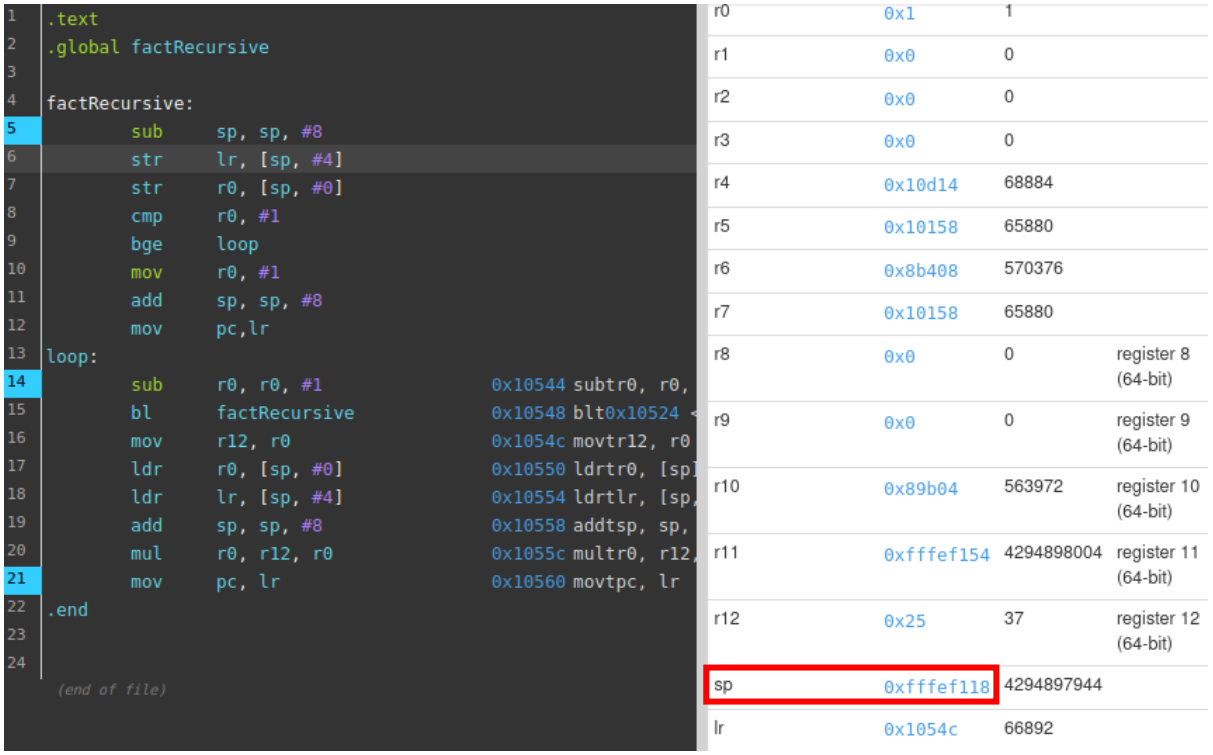
registers

name	value (hex)	value (decimal)	description
r0	0x1	1	
r1	0x0	0	
r2	0x0	0	
r3	0x0	0	
r4	0x10d14	68884	
r5	0x10158	65880	
r6	0x8b408	570376	
r7	0x10158	65880	
r8	0x0	0	register 8 (64-bit)
r9	0x0	0	register 9 (64-bit)
r10	0x89b04	563972	register 10 (64-bit)

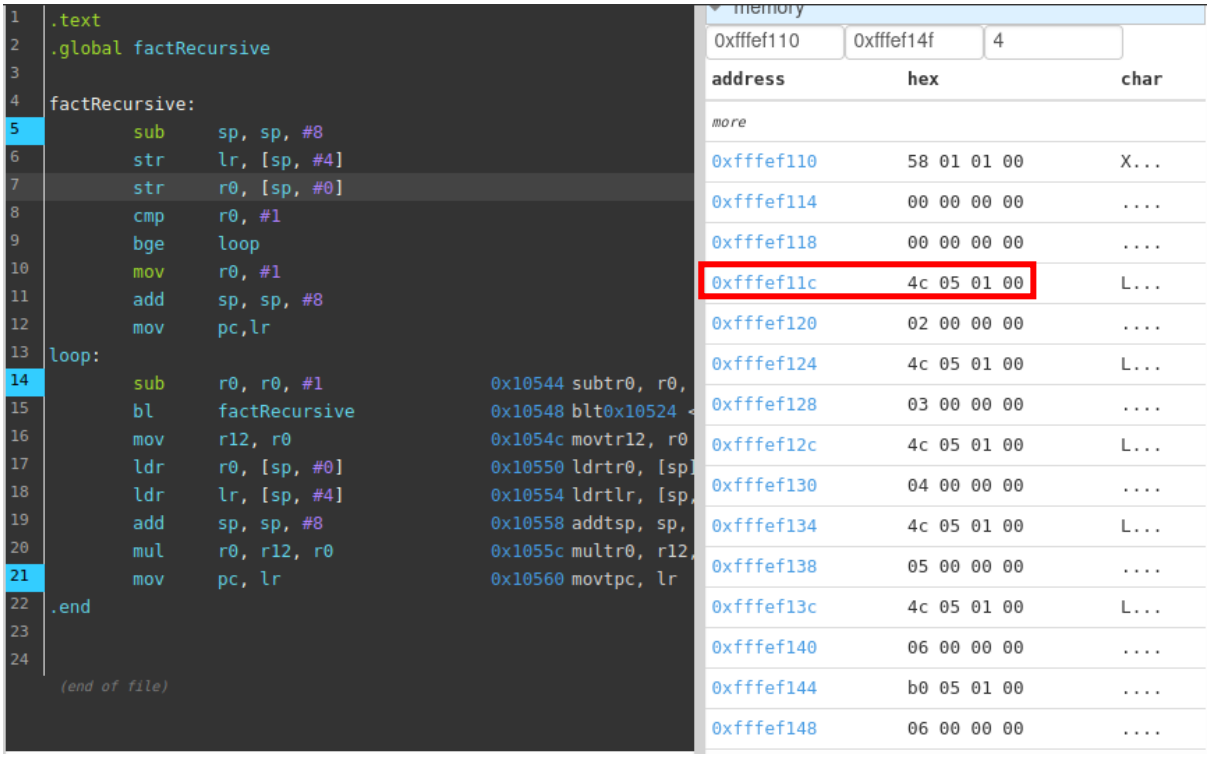
sub r0,r0,#1 명령어 수행 -> r0의 값을 1만큼 감소시켰다. 이제 r0는 1이다.



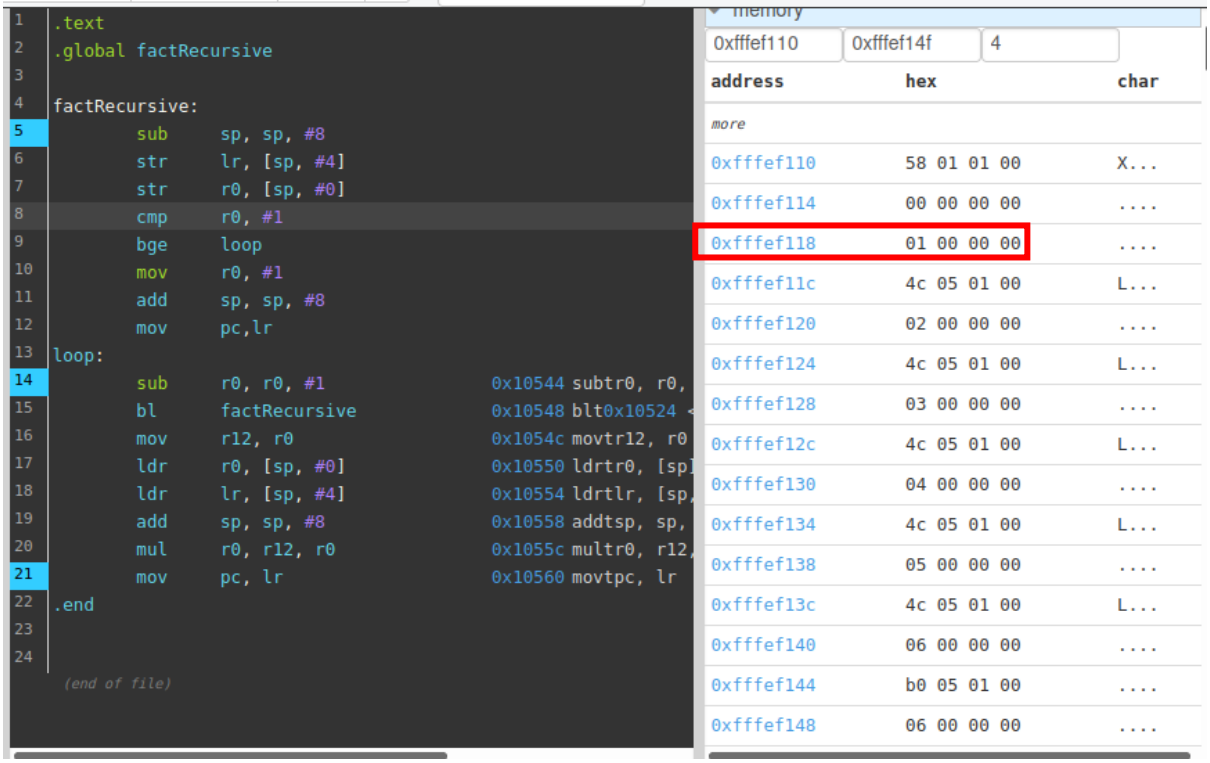
bl factRecursive 명령어 수행 -> branch 명령에 따라 factRecursive로 branch한다. link register에 다음 명령(mov r12,r0)의 pc값 0x1054c를 저장한다.



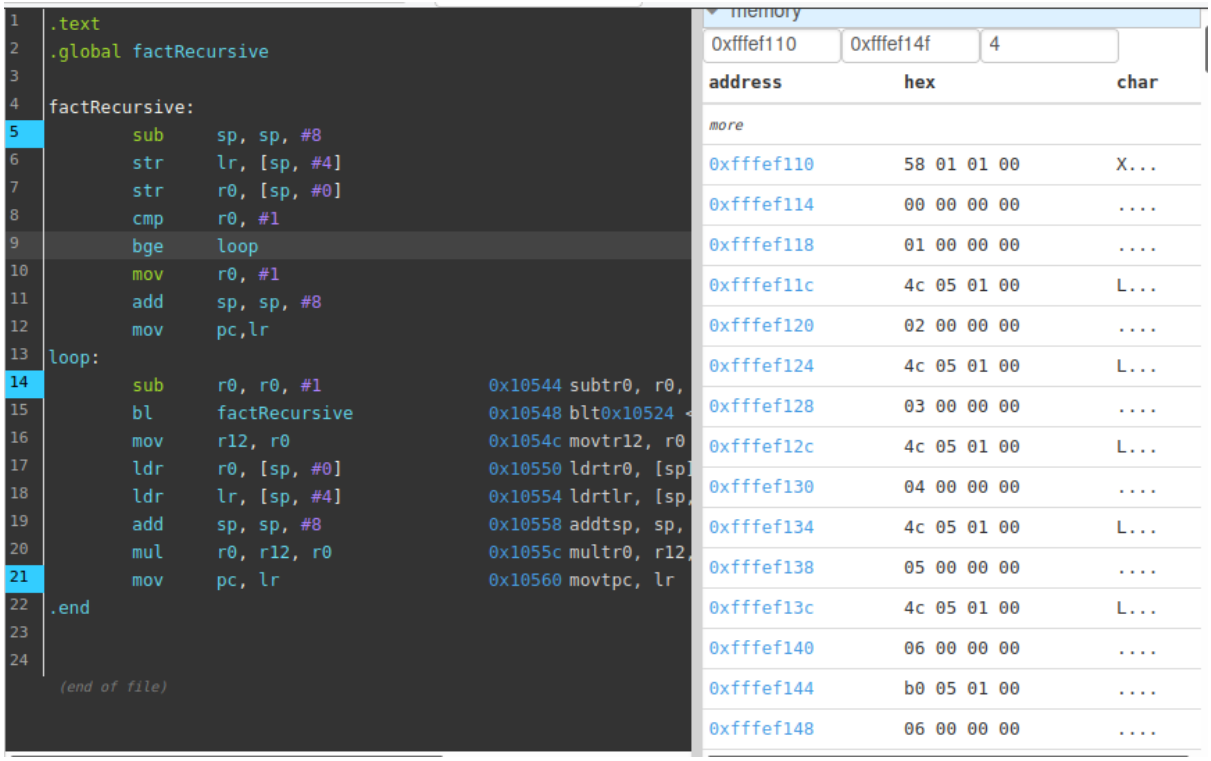
sub sp,sp,#8 명령어 수행 -> 기존 sp값 0xfffef120에서 8만큼 감소시켰다.



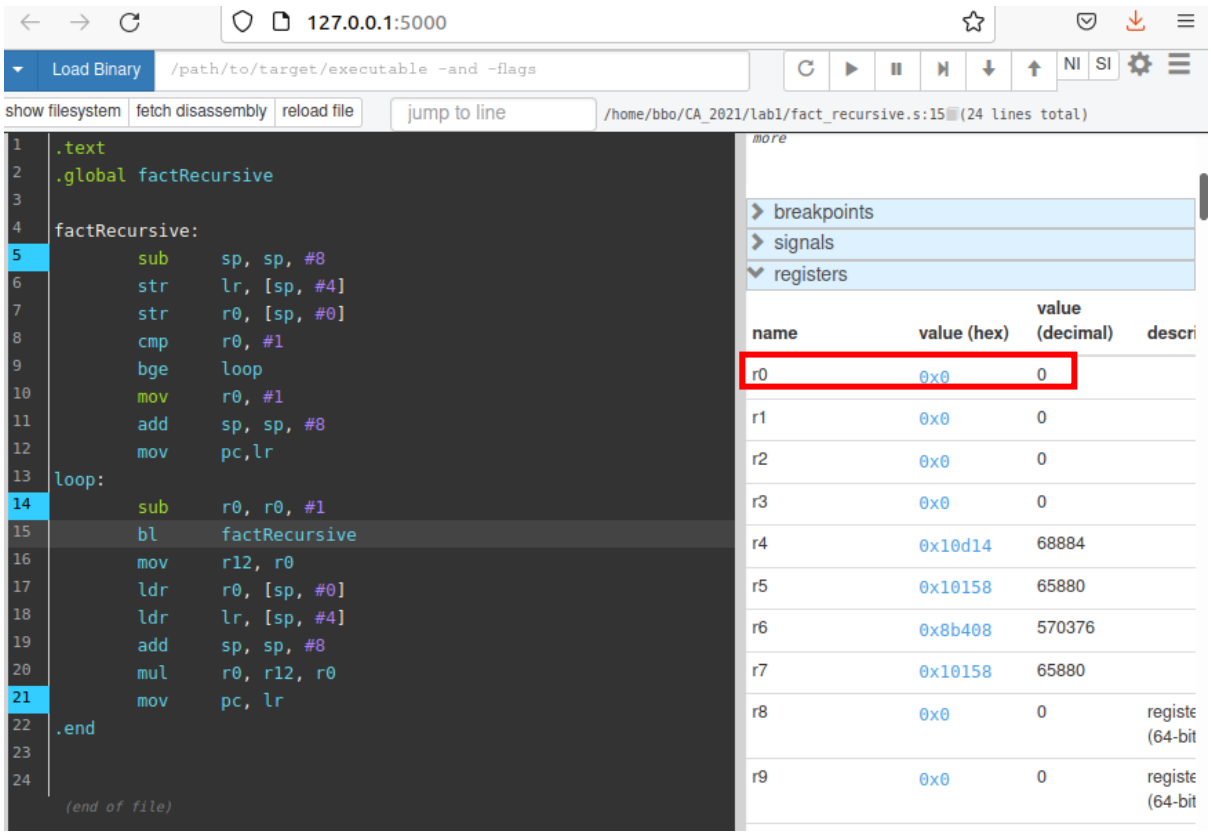
str lr, [sp, #4] 명령어 수행 -> 현재 sp값 + 4의 위치에 link register 값을 저장했다.



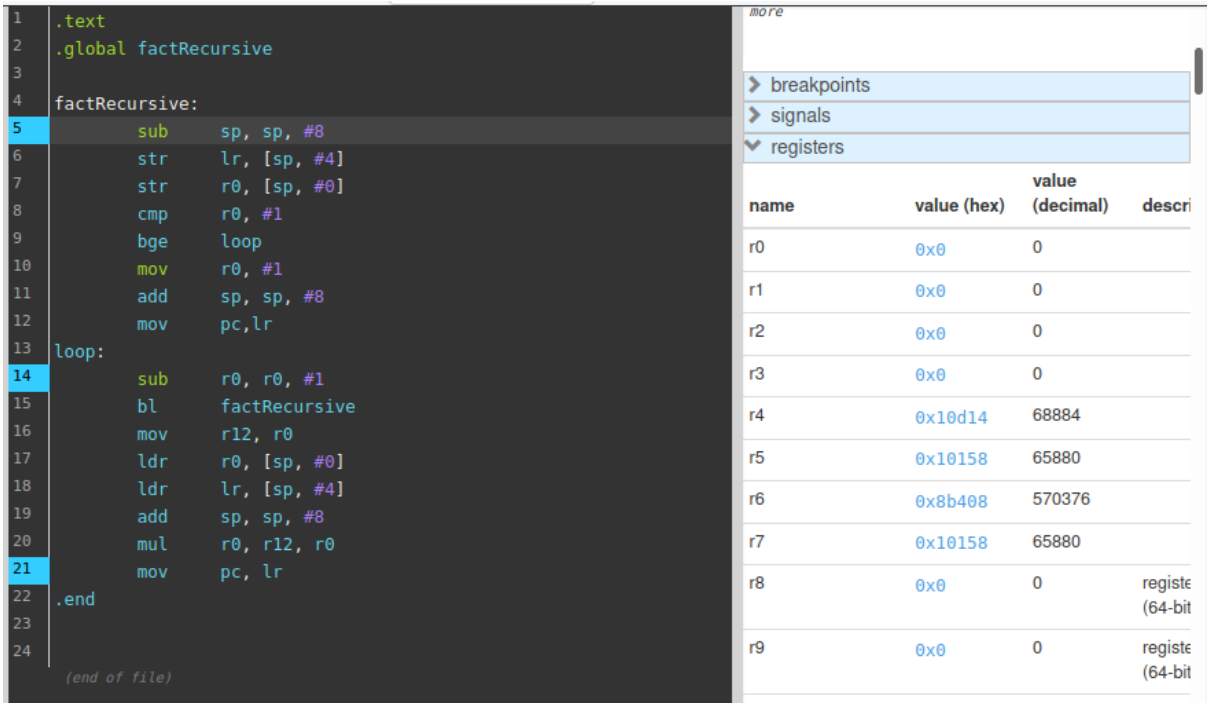
str r0, [sp, #0] 명령어 수행 -> 현재 sp값의 위치에 r0(1)을 저장했다.



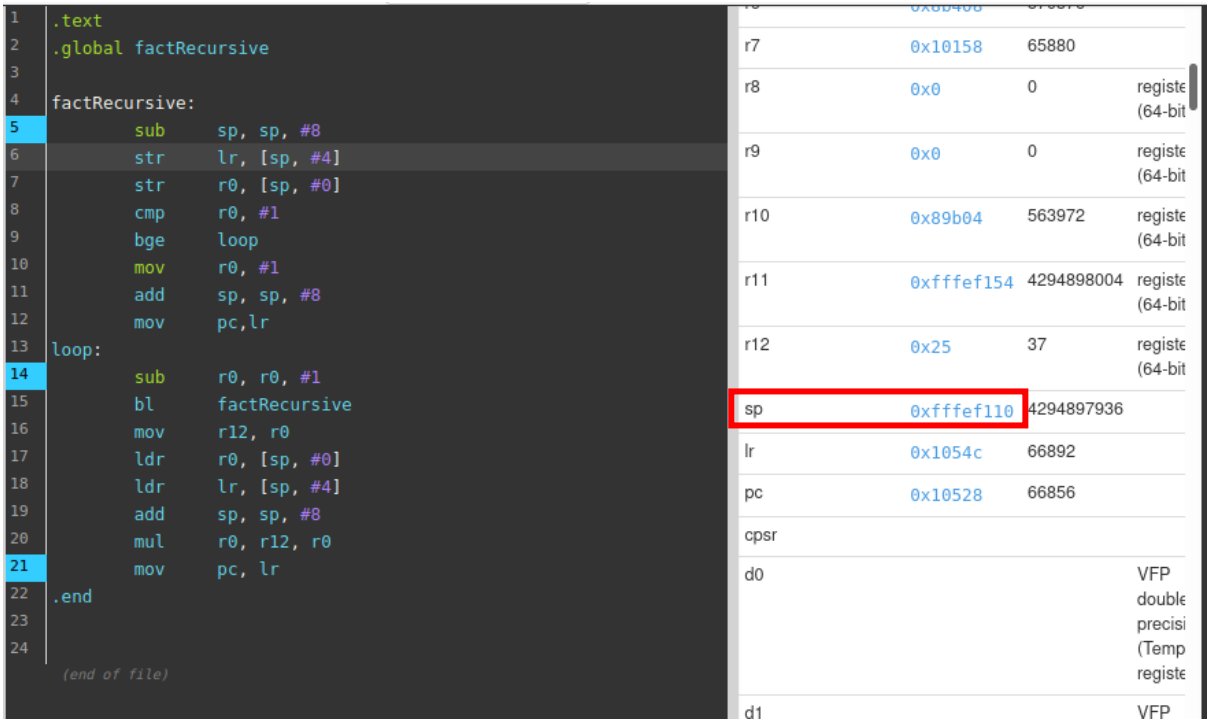
cmp r0, #1과 bge loop 명령어 수행 -> r0와 1의 값을 비교, r0값이 같으므로 loop로 branch한다.



sub r0,r0,#1 명령어 수행 -> r0의 값을 1만큼 감소시켰다. 이제 r0는 0이다.



bl factRecursive 명령어 수행 -> branch 명령에 따라 factRecursive로 branch한다. link register에 다음 명령(mov r12,r0)의 pc값 0x1054c를 저장한다.



sub sp,sp,#8 명령어 수행 -> 기존 sp값 0xfffef118에서 8만큼 감소시켰다.

```
1  .text
2  .global factRecursive
3
4  factRecursive:
5          sub     sp, sp, #8
6          str     lr, [sp, #4]
7          str     r0, [sp, #0]
8          cmp     r0, #1
9          bge     loop
10         mov     r0, #1
11         add     sp, sp, #8
12         mov     pc,lr
13 loop:
14         sub     r0, r0, #1
15         bl      factRecursive
16         mov     r12, r0
17         ldr     r0, [sp, #0]
18         ldr     lr, [sp, #4]
19         add     sp, sp, #8
20         mul     r0, r12, r0
21         mov     pc, lr
22 .end
23
24
    (end of file)
```

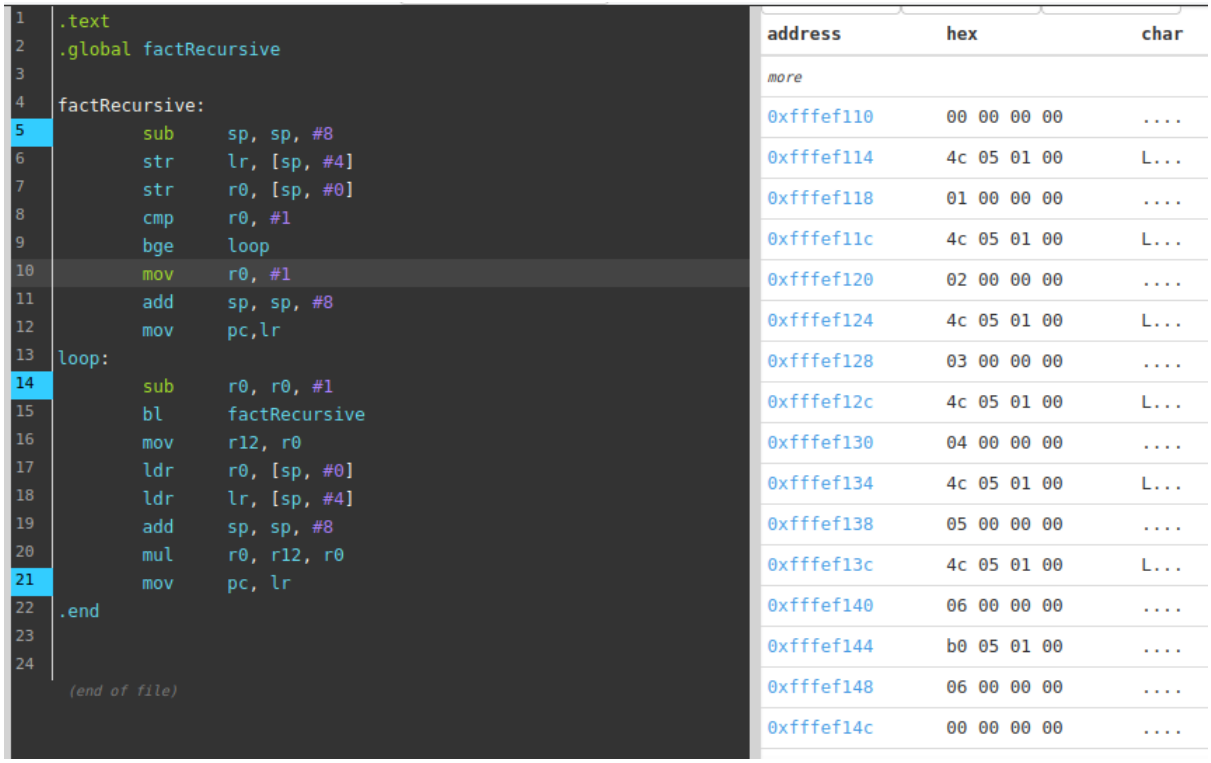
address	hex	char
more		
0xfffef110	58 01 01 00	X...
0xfffef114	4c 05 01 00	L...
0xfffef118	01 00 00 00	....
0xfffef11c	4c 05 01 00	L...
0xfffef120	02 00 00 00	....
0xfffef124	4c 05 01 00	L...
0xfffef128	03 00 00 00	....
0xfffef12c	4c 05 01 00	L...
0xfffef130	04 00 00 00	....
0xfffef134	4c 05 01 00	L...
0xfffef138	05 00 00 00	....
0xfffef13c	4c 05 01 00	L...
0xfffef140	06 00 00 00	....
0xfffef144	b0 05 01 00	....
0xfffef148	06 00 00 00	....

str lr, [sp, #4] 명령어 수행 -> 현재 sp값 + 4의 위치에 link register 값을 저장했다.

```
1  .text
2  .global factRecursive
3
4  factRecursive:
5          sub     sp, sp, #8
6          str     lr, [sp, #4]
7          str     r0, [sp, #0]
8          cmp     r0, #1
9          bge     loop
10         mov     r0, #1
11         add     sp, sp, #8
12         mov     pc,lr
13 loop:
14         sub     r0, r0, #1
15         bl      factRecursive
16         mov     r12, r0
17         ldr     r0, [sp, #0]
18         ldr     lr, [sp, #4]
19         add     sp, sp, #8
20         mul     r0, r12, r0
21         mov     pc, lr
22 .end
23
24
    (end of file)
```

address	hex	char
more		
0xfffef110	00 00 00 00	....
0xfffef114	4c 05 01 00	L...
0xfffef118	01 00 00 00	....
0xfffef11c	4c 05 01 00	L...
0xfffef120	02 00 00 00	....
0xfffef124	4c 05 01 00	L...
0xfffef128	03 00 00 00	....
0xfffef12c	4c 05 01 00	L...
0xfffef130	04 00 00 00	....
0xfffef134	4c 05 01 00	L...
0xfffef138	05 00 00 00	....
0xfffef13c	4c 05 01 00	L...
0xfffef140	06 00 00 00	....
0xfffef144	b0 05 01 00	....
0xfffef148	06 00 00 00	....
0xfffef14c	00 00 00 00	....

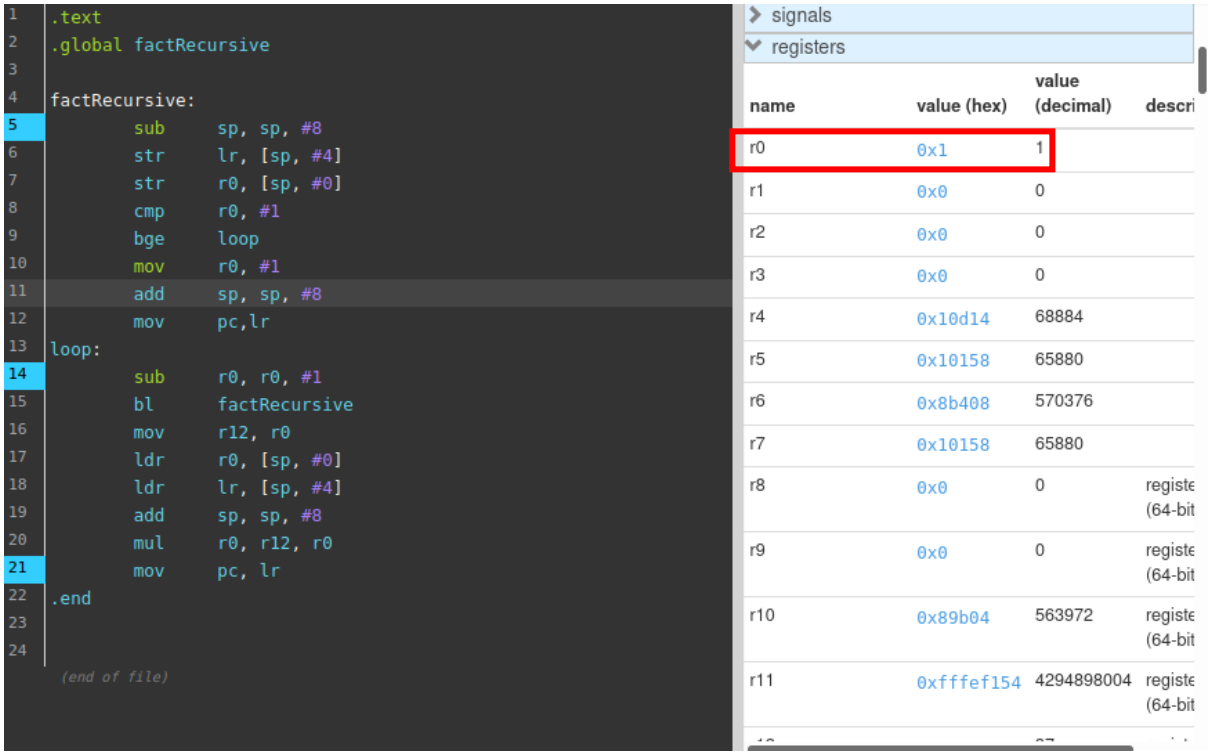
str r0, [sp, #0] 명령어 수행 -> 현재 sp값의 위치에 r0(0)을 저장했다.



```
1  .text
2  .global factRecursive
3
4  factRecursive:
5          sub     sp, sp, #8
6          str     lr, [sp, #4]
7          str     r0, [sp, #0]
8          cmp     r0, #1
9          bge     loop
10         mov     r0, #1
11         add     sp, sp, #8
12         mov     pc,lr
13 loop:
14         sub     r0, r0, #1
15         bl      factRecursive
16         mov     r12, r0
17         ldr     r0, [sp, #0]
18         ldr     lr, [sp, #4]
19         add     sp, sp, #8
20         mul     r0, r12, r0
21         mov     pc, lr
22 .end
23
24
   (end of file)
```

address	hex	char
more		
0xfffef110	00 00 00 00	....
0xfffef114	4c 05 01 00	L...
0xfffef118	01 00 00 00	....
0xfffef11c	4c 05 01 00	L...
0xfffef120	02 00 00 00	....
0xfffef124	4c 05 01 00	L...
0xfffef128	03 00 00 00	....
0xfffef12c	4c 05 01 00	L...
0xfffef130	04 00 00 00	....
0xfffef134	4c 05 01 00	L...
0xfffef138	05 00 00 00	....
0xfffef13c	4c 05 01 00	L...
0xfffef140	06 00 00 00	....
0xfffef144	b0 05 01 00	....
0xfffef148	06 00 00 00	....
0xfffef14c	00 00 00 00	....

cmp r0, #1과 bge loop 명령어 수행 -> r0와 1의 값을 비교, r0값이 0으로 1보다 작으므로 branch 하지 않고 다음 명령어 mov r0, #1로 이동한다.



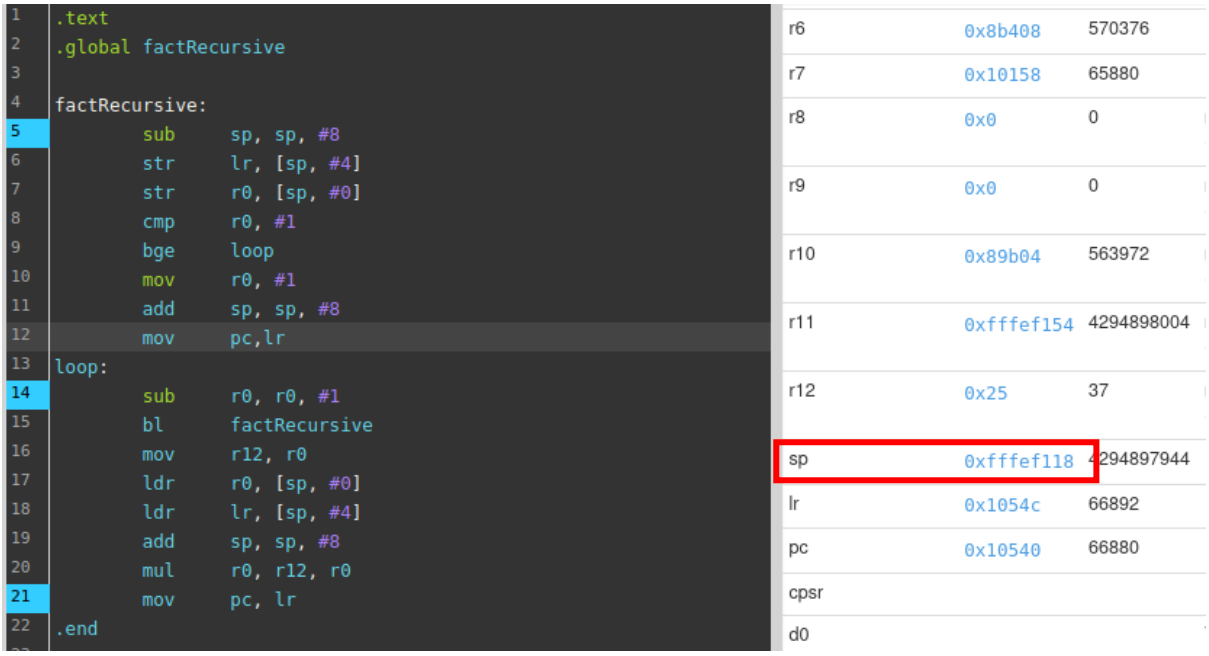
```
1  .text
2  .global factRecursive
3
4  factRecursive:
5          sub     sp, sp, #8
6          str     lr, [sp, #4]
7          str     r0, [sp, #0]
8          cmp     r0, #1
9          bge     loop
10         mov     r0, #1
11         add     sp, sp, #8
12         mov     pc,lr
13 loop:
14         sub     r0, r0, #1
15         bl      factRecursive
16         mov     r12, r0
17         ldr     r0, [sp, #0]
18         ldr     lr, [sp, #4]
19         add     sp, sp, #8
20         mul     r0, r12, r0
21         mov     pc, lr
22 .end
23
24
   (end of file)
```

signals

registers

name	value (hex)	value (decimal)	descri
r0	0x1	1	
r1	0x0	0	
r2	0x0	0	
r3	0x0	0	
r4	0x10d14	68884	
r5	0x10158	65880	
r6	0x8b408	570376	
r7	0x10158	65880	
r8	0x0	0	registe (64-bit
r9	0x0	0	registe (64-bit
r10	0x89b04	563972	registe (64-bit
r11	0xfffef154	4294898004	registe (64-bit

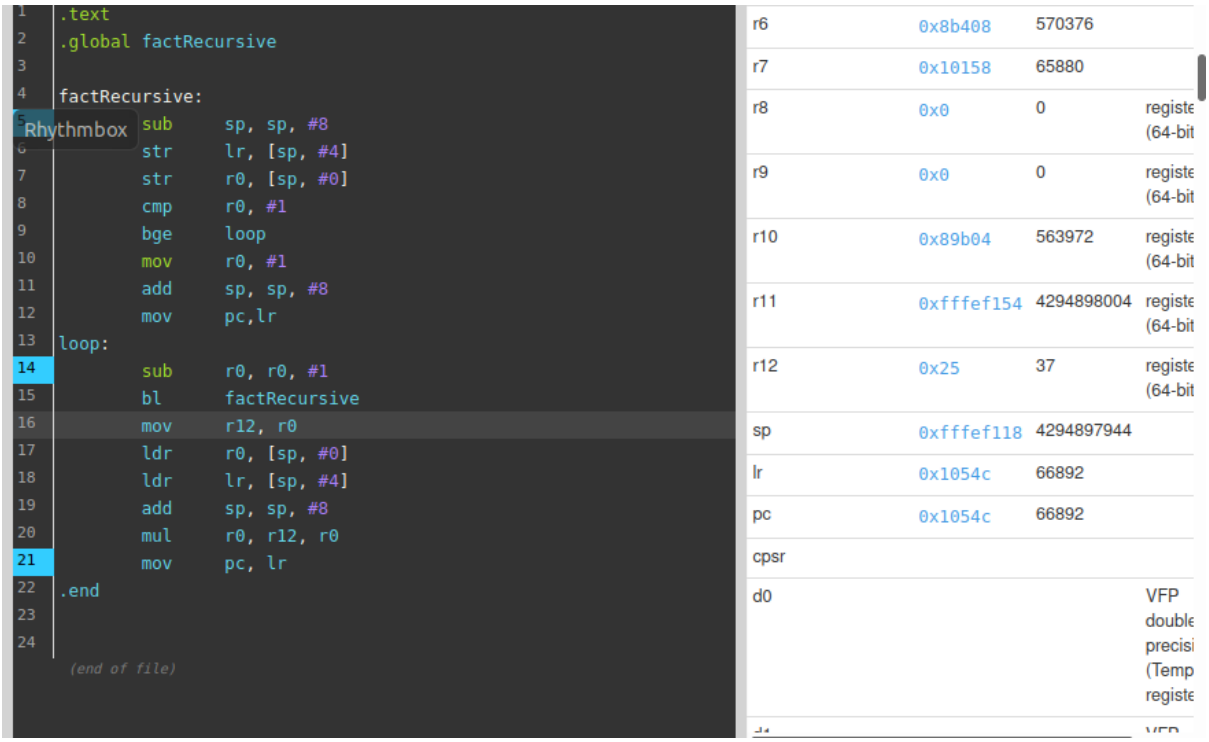
mov r0, #1 명령어 수행 -> r0에 1의 값을 넣었다. 이제 r0는 1이다.



```
.text
.global factRecursive

factRecursive:
        sub     sp, sp, #8
        str     lr, [sp, #4]
        str     r0, [sp, #0]
        cmp     r0, #1
        bge     loop
        mov     r0, #1
        add     sp, sp, #8
        mov     pc,lr
loop:
        sub     r0, r0, #1
        bl      factRecursive
        mov     r12, r0
        ldr     r0, [sp, #0]
        ldr     lr, [sp, #4]
        add     sp, sp, #8
        mul     r0, r12, r0
        mov     pc, lr
.end
```

r6	0x8b408	570376
r7	0x10158	65880
r8	0x0	0
r9	0x0	0
r10	0x89b04	563972
r11	0xfffef154	4294898004
r12	0x25	37
sp	0xfffef118	4294897944
lr	0x1054c	66892
pc	0x10540	66880
cpsr		
d0		

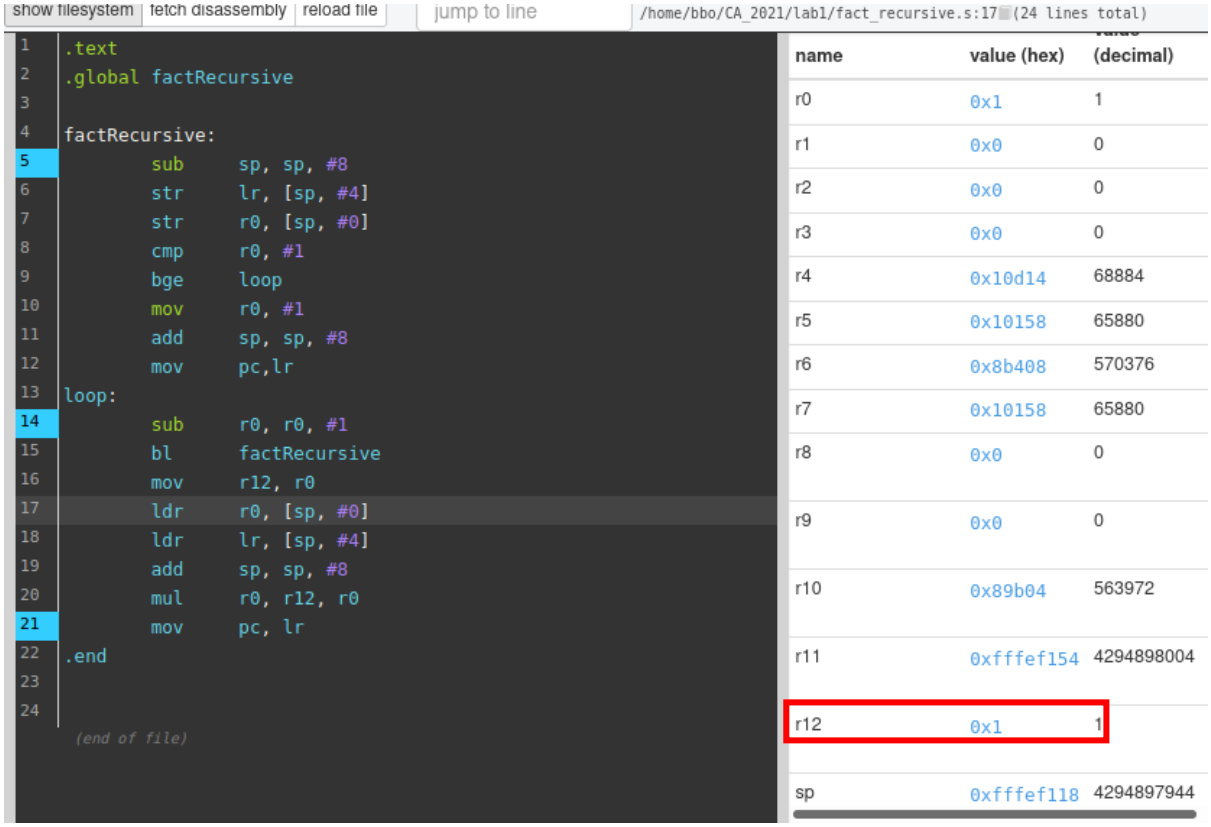
add sp, sp, #8 명령어 수행 -> sp에 8을 더한다.(pop동작)



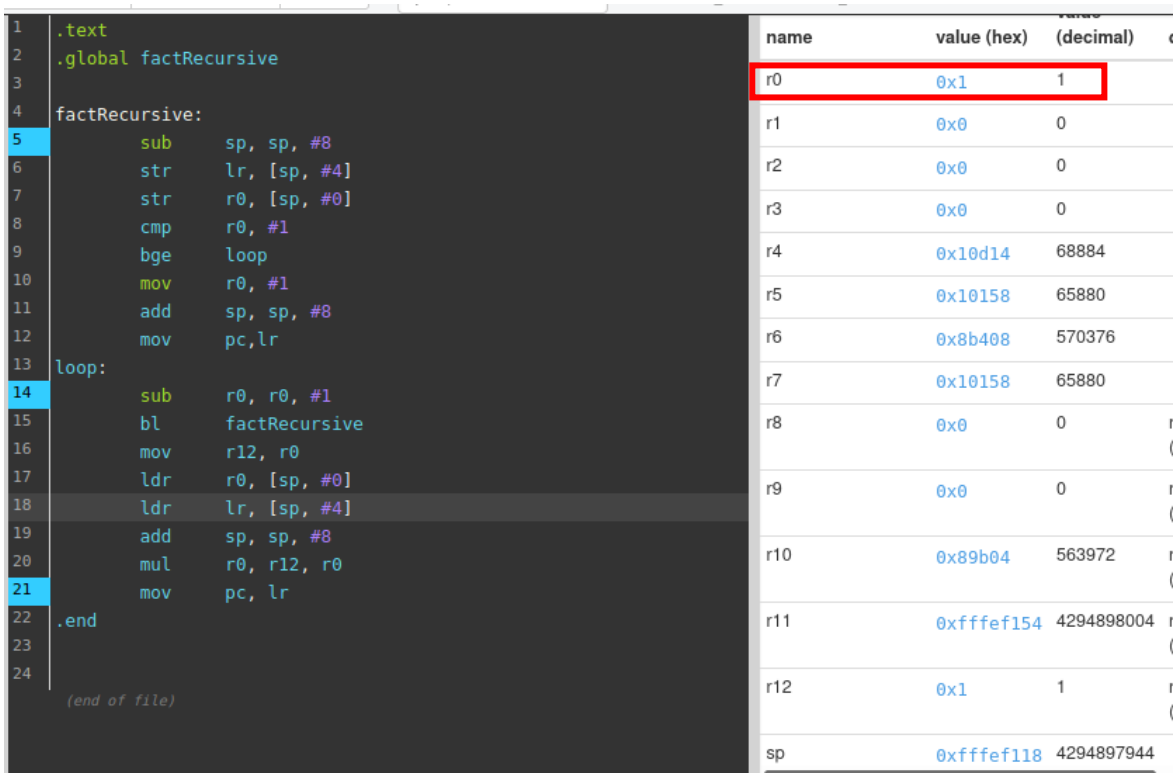
```
.text
.global factRecursive

factRecursive:
        sub     sp, sp, #8
        str     lr, [sp, #4]
        str     r0, [sp, #0]
        cmp     r0, #1
        bge     loop
        mov     r0, #1
        add     sp, sp, #8
        mov     pc,lr
loop:
        sub     r0, r0, #1
        bl      factRecursive
        mov     r12, r0
        ldr     r0, [sp, #0]
        ldr     lr, [sp, #4]
        add     sp, sp, #8
        mul     r0, r12, r0
        mov     pc, lr
.end


(end of file)
```

r6	0x8b408	570376	
r7	0x10158	65880	
r8	0x0	0	registe (64-bit
r9	0x0	0	registe (64-bit
r10	0x89b04	563972	registe (64-bit
r11	0xfffef154	4294898004	registe (64-bit
r12	0x25	37	registe (64-bit
sp	0xfffef118	4294897944	
lr	0x1054c	66892	
pc	0x1054c	66892	
cpsr			
d0			VFP double precisi (Temp registe

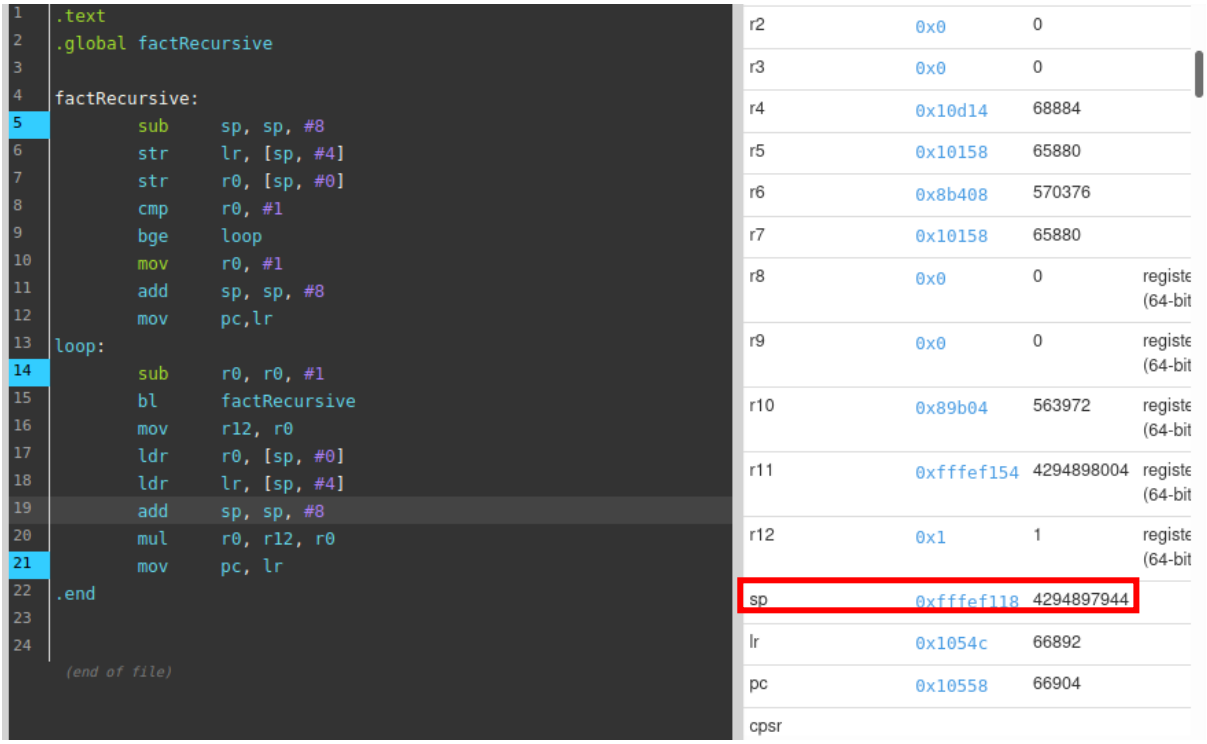
mov pc, lr 명령어 수행 -> program counter 값을 link register의 값으로 교체한다. link register에는 mov r12, r0의 pc값이 저장되어있다.



mov r12, r0 명령어 수행 -> r12에 r0의 값을 넣는다. r12는 이제 1이다.



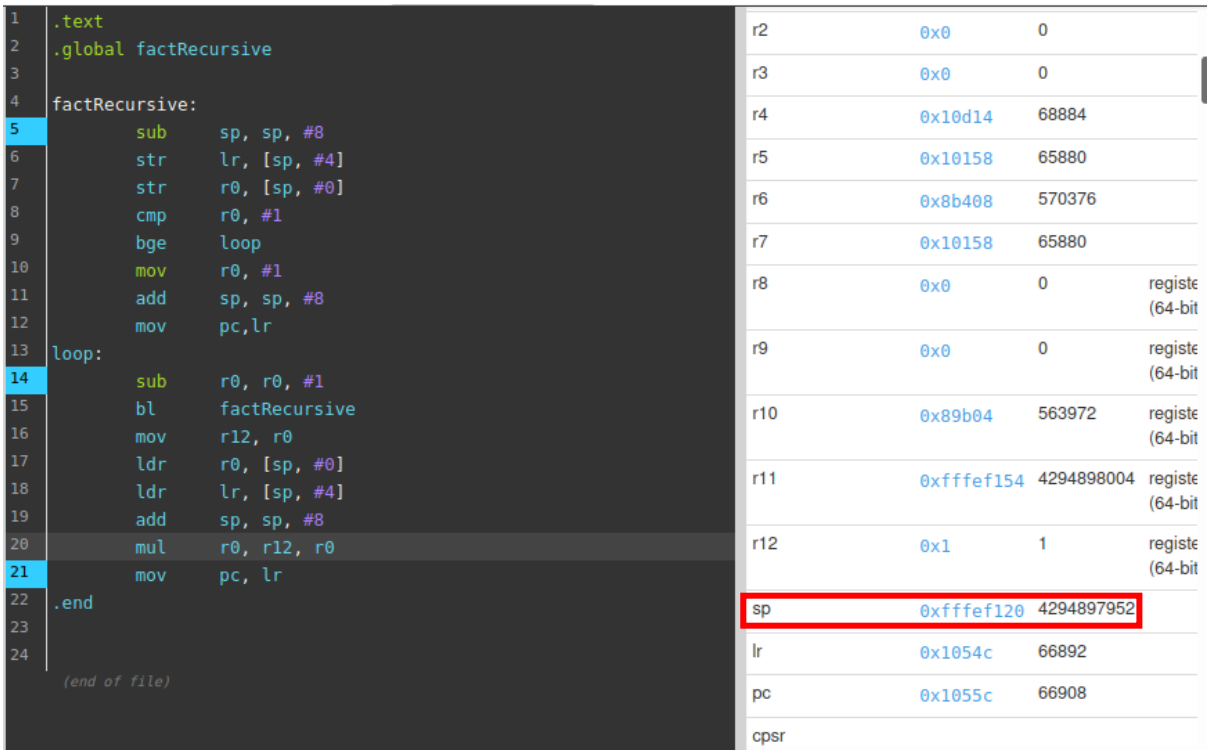
ldr r0, [sp, #0] 명령어 수행 -> sp에 위치한 값을 r0에 저장한다. 현재 0xfffef118에는 1이 저장되어있다. 따라서 r0는 1이 된다.



```
.text
.global factRecursive

factRecursive:
        sub     sp, sp, #8
        str     lr, [sp, #4]
        str     r0, [sp, #0]
        cmp     r0, #1
        bge     loop
        mov     r0, #1
        add     sp, sp, #8
        mov     pc,lr
loop:
        sub     r0, r0, #1
        bl      factRecursive
        mov     r12, r0
        ldr     r0, [sp, #0]
        ldr     lr, [sp, #4]
        add     sp, sp, #8
        mul     r0, r12, r0
        mov     pc, lr
.end

(end of file)
```

r2	0x0	0	
r3	0x0	0	
r4	0x10d14	68884	
r5	0x10158	65880	
r6	0x8b408	570376	
r7	0x10158	65880	
r8	0x0	0	registe (64-bit
r9	0x0	0	registe (64-bit
r10	0x89b04	563972	registe (64-bit
r11	0xfffef154	4294898004	registe (64-bit
r12	0x1	1	registe (64-bit
sp	0xfffef118	4294897944	
lr	0x1054c	66892	
pc	0x10558	66904	
cpsr			

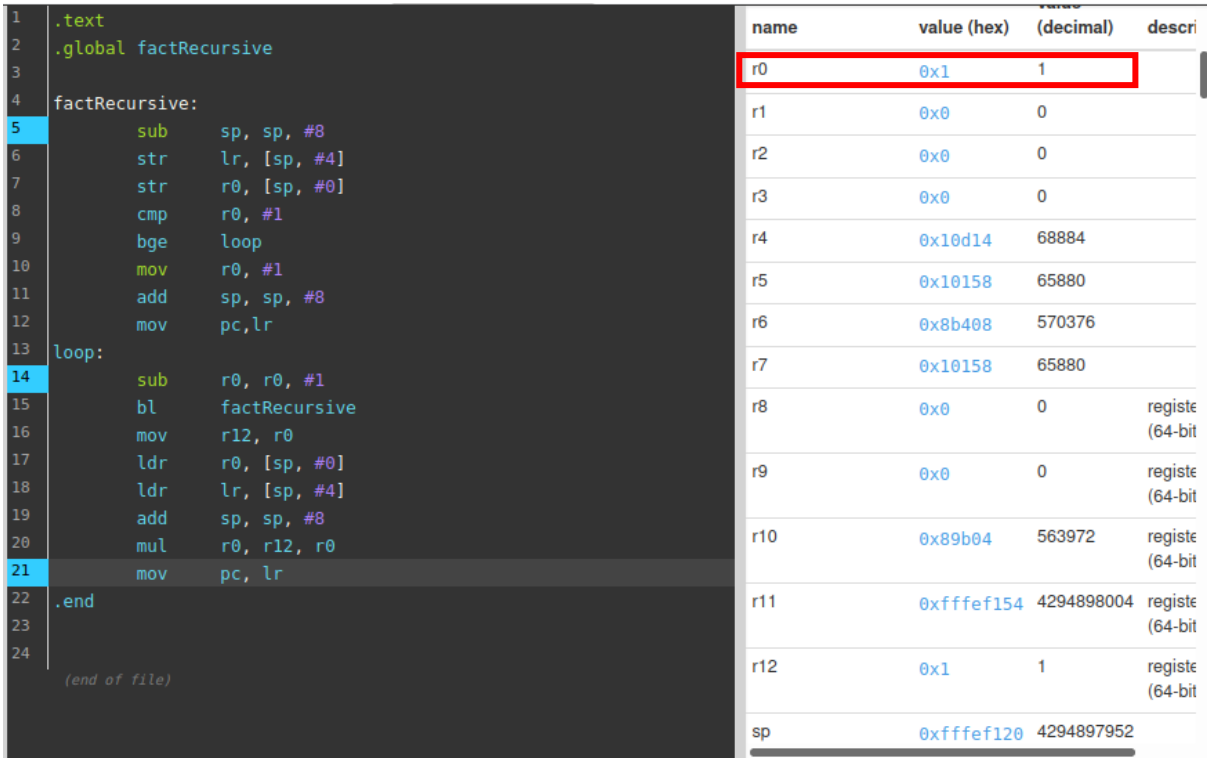
ldr r0, [sp, #4] 명령어 수행 -> sp + 4에 위치한 값을 lr에 저장한다. 현재 0xfffef11c에는 0x1054c가 저장되어있다. 따라서 lr은 0x1054c가 된다.



```
.text
.global factRecursive

factRecursive:
        sub     sp, sp, #8
        str     lr, [sp, #4]
        str     r0, [sp, #0]
        cmp     r0, #1
        bge     loop
        mov     r0, #1
        add     sp, sp, #8
        mov     pc,lr
loop:
        sub     r0, r0, #1
        bl      factRecursive
        mov     r12, r0
        ldr     r0, [sp, #0]
        ldr     lr, [sp, #4]
        add     sp, sp, #8
        mul     r0, r12, r0
        mov     pc, lr
.end

(end of file)
```

r2	0x0	0	
r3	0x0	0	
r4	0x10d14	68884	
r5	0x10158	65880	
r6	0x8b408	570376	
r7	0x10158	65880	
r8	0x0	0	registe (64-bit
r9	0x0	0	registe (64-bit
r10	0x89b04	563972	registe (64-bit
r11	0xfffef154	4294898004	registe (64-bit
r12	0x1	1	registe (64-bit
sp	0xfffef120	4294897952	
lr	0x1054c	66892	
pc	0x1055c	66908	
cpsr			

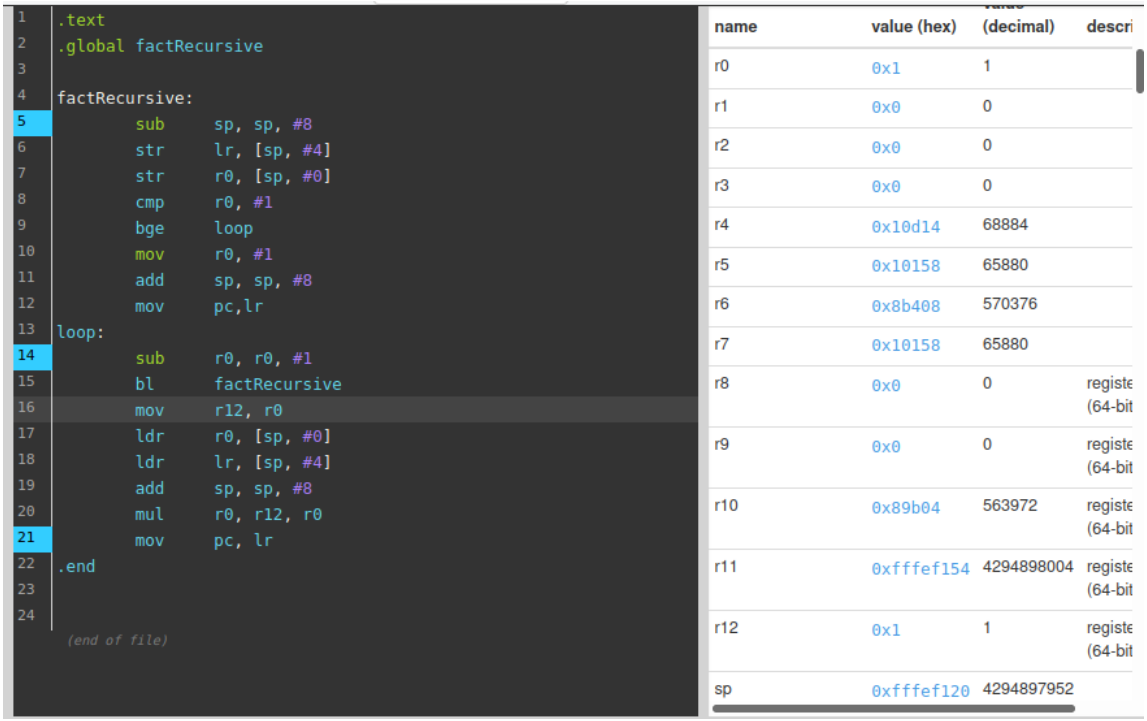
add sp, sp, #8 명령어 수행 -> sp에 8을 더한다. (pop동작)



```
1  .text
2  .global factRecursive
3
4  factRecursive:
5          sub     sp, sp, #8
6          str     lr, [sp, #4]
7          str     r0, [sp, #0]
8          cmp     r0, #1
9          bge     loop
10         mov     r0, #1
11         add     sp, sp, #8
12         mov     pc,lr
13 loop:
14         sub     r0, r0, #1
15         bl      factRecursive
16         mov     r12, r0
17         ldr     r0, [sp, #0]
18         ldr     lr, [sp, #4]
19         add     sp, sp, #8
20         mul     r0, r12, r0
21         mov     pc, lr
22 .end
23
24
   (end of file)
```

name	value (hex)	(decimal)	descri
r0	0x1	1	
r1	0x0	0	
r2	0x0	0	
r3	0x0	0	
r4	0x10d14	68884	
r5	0x10158	65880	
r6	0x8b408	570376	
r7	0x10158	65880	
r8	0x0	0	registe (64-bit
r9	0x0	0	registe (64-bit
r10	0x89b04	563972	registe (64-bit
r11	0xfffef154	4294898004	registe (64-bit
r12	0x1	1	registe (64-bit
sp	0xfffef120	4294897952	

mul r0, r12, r0 명령어 수행 -> r0에 r0와 r12를 곱한 값을 저장한다. r0가 1, r12가 1이므로 r0에 1이 저장된다.



```
1  .text
2  .global factRecursive
3
4  factRecursive:
5          sub     sp, sp, #8
6          str     lr, [sp, #4]
7          str     r0, [sp, #0]
8          cmp     r0, #1
9          bge     loop
10         mov     r0, #1
11         add     sp, sp, #8
12         mov     pc,lr
13 loop:
14         sub     r0, r0, #1
15         bl      factRecursive
16         mov     r12, r0
17         ldr     r0, [sp, #0]
18         ldr     lr, [sp, #4]
19         add     sp, sp, #8
20         mul     r0, r12, r0
21         mov     pc, lr
22 .end
23
24
   (end of file)
```

name	value (hex)	(decimal)	descri
r0	0x1	1	
r1	0x0	0	
r2	0x0	0	
r3	0x0	0	
r4	0x10d14	68884	
r5	0x10158	65880	
r6	0x8b408	570376	
r7	0x10158	65880	
r8	0x0	0	registe (64-bit
r9	0x0	0	registe (64-bit
r10	0x89b04	563972	registe (64-bit
r11	0xfffef154	4294898004	registe (64-bit
r12	0x1	1	registe (64-bit
sp	0xfffef120	4294897952	

mov pc, lr 명령어 수행 -> program counter 값을 link register의 값으로 교체한다. link register에는 mov r12, r0의 pc값이 저장되어있다. 이를 반복하다보면 r0가 720(r12에 6이 저장된후 계산)되고, lr에 저장된 값에 따라 main의 fact함수로 결과 값을 가지고 복귀한다.

2) 실습 B

8, 9번코드에서 **cmp r0, #1, bge loop** 명령어를 수행한다.

bge는 greater or equal, 크거나 같으면 branch 시킨다.

일반적인 경우 **N=V일 때, bge의 조건이 만족되어 branch하게 된다.**

① r0가 2 이상의 양수일 때, sub r0, #1에서 N = 0, V = 0, Z = 0, C = 0 으로 branch한다.

② r0가 1일 때, sub r0, #1에서 N = 0, V = 0, Z = 1, C = 0 으로 branch한다.

③ r0가 0 이하의 정수일 때, sub r0, #1에서 N = 1, V = 0, Z = 0, C = 0 으로 branch하지 않고 다음 명령어로 이동한다.

또한 operand2가 1로 고정되어 있으므로, overflow가 발생할 경우는 하나밖에 없다.

① r0가 음수이고 sub r0, #1에서 overflow발생한다. N = 0, V = 1, Z = 0, C = 1 이므로 branch하지 않고 다음 명령어로 이동한다.